Java Exceptions

Version 2.1 - May2013

Licensing Note

Motivation

- Report errors, by delegating error handling to higher levels
- Callee might not know how to recover from an error
- Caller of a method can handle error in a more appropriate way than the callee

- Localize error handling code, by separating it from functional code
- Functional code is more readable
- Error code is centralized, rather than being scattered

The world without exceptions (I)

- If a non locally remediable error happens while method is executing, call `System.exit()`

- A method causing an unconditional program interruption in not very dependable (nor usable)

The world without exceptions (II)

- If errors happen while method is executing, we return a special value
- Special values are different from normal return value (e.g., null, -1, etc.)

- Developer must remember value/meaning of special values for each call to check for errors
- What if all values are normal?
  - ♦ double pow(base, exponent)
  - ♦ pow(-1, 0.5); //not a real

Real problems

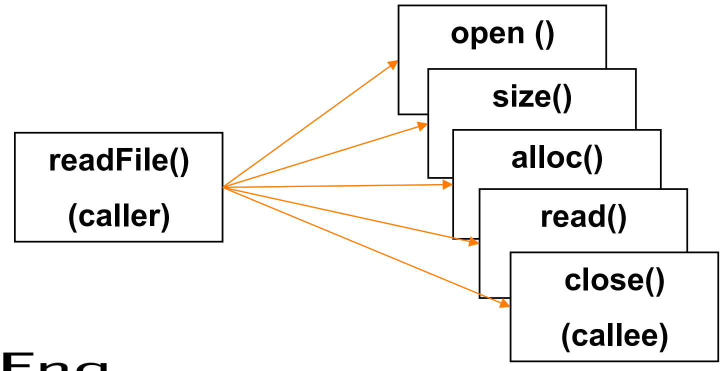
- Code is messier to write and harder to read

```
if( somefunc() == ERROR ) // detect error
   //handle the error
else
   //proceed normally
```

- Only the direct caller can intercept errors (no delegation to any upward method)

Example – Read file

- open the file
- determine file size
- allocate that much memory
- read the file into memory
- close the file



readFile()
(caller)

open ()

size()

alloc()

read()

close()
(callee)

All of them can fail

Correct (but boring)

```
int readFile {
   open the file;
   if (operationFailed)
      return -1;
   determine file size;
   if (operationFailed)
      return -2;
   allocate that much memory;
   if (operationFailed) {
      close the file;
      return -3;
   }
   read the file into memory;
   if (operationFailed) {
      close the file;
      return -4;
   }
   close the file;
   if (operationFailed)
      return -5;
   return 0;
}
```

Lots of error-detection and error-handling code

To detect errors we must check specs of library calls (no homogeneity)

Unreadable

Wrong (but quick and readable)

```
int readFile {

   open the file;
   determine file size;
   allocate that much memory;
   read the file into memory;
   close the file;

   return 0;
}
```

Which one would YOU use ?

☺

Using exceptions (nice)

```
try {
        open the file;
        determine file size;
        allocate that much memory;
        read the file into memory;
        close the file;
} catch (fileOpenFailed) {
        doSomething;
} catch (sizeDeterminationFailed) {
        doSomething;
} catch (memoryAllocationFailed) {
        doSomething;
} catch (readFailed) {
        doSomething;
} catch (fileCloseFailed) {
        doSomething;
}
```

Basic concepts

- The code causing the error will generate an exception
  - Developers code
  - Third-party library
- At some point up in the hierarchy of method invocations, a caller will intercept and stop the exception
- In between, methods can
  - Ignore the exception (complete delegation)
  - Intercept without stopping (partial delegation)

Syntax

- Java provides three keywords
  - Throw
    - Generates an exception
  - Try
    - Contains code that may generate exceptions
  - Catch
    - Defines the error handler
- We also need a new entity
  - Exception class

Generation

1. Declare an exception class
2. Mark the method generating the exception
3. Create an exception object
4. Throw upward the exception

Generation

```
// java.lang.Exception                               (1)
public class EmptyStack extends Exception {
}

class Stack{                                        (2)
   public Object pop() throws EmptyStack {

      if(size == 0) {                                (3)
         Exception e = new EmptyStack();
         throw e;
      }                  (4)
      ...
   }
}
```

throws

- Method interface must declare exception type(s) generated within its implementation (list with commas)

- Either generated and thrown by method, directly
- Or generated by other methods called within the method and not caught

throw

- Execution of current method is interrupted immediatelly
- Catching phase starts

Interception

- Catching exceptions generated in a code portion

```
try {
   // in this piece of code some
   // exceptions may be generated
   stack.pop();
   ...
}
catch (StackEmpty e) {

   // error handling
   System.out.println(e);
   ...
}
```

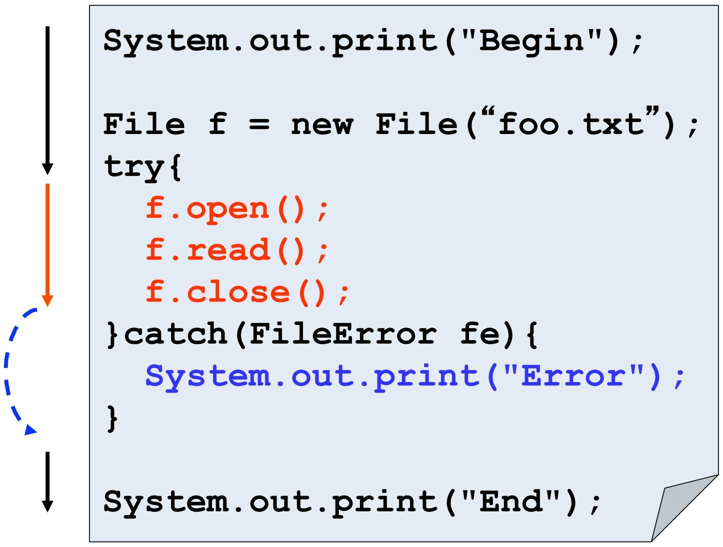
Execution flow

- open and close can generate a FileError
- Suppose read does not generate exceptions

```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
   f.open();
   f.read();
   f.close();
}catch(FileError fe){
   System.out.print("Error");
}

System.out.print("End");
```

Execution flow

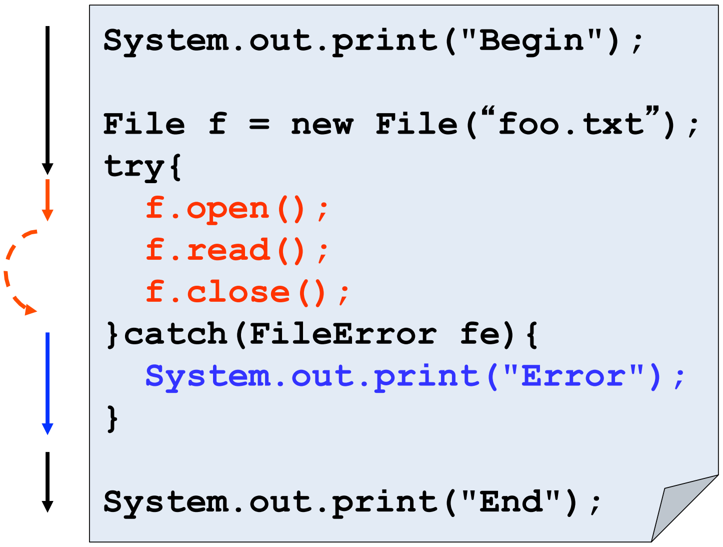
- No exception generated



```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
   f.open();
   f.read();
   f.close();
}catch(FileError fe){
   System.out.print("Error");
}

System.out.print("End");
```

Execution flow

- open() generates an exception
- read() and close() are skipped



```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
   f.open();
   f.read();
   f.close();
}catch(FileError fe){
   System.out.print("Error");
}

System.out.print("End");
```

Multiple catch

- Capturing different types of exception is possible with different catch blocks

```
try {
  ...
}
catch(StackEmpty se) {
   // here stack errors are handled
}
catch(IOException ioe) {
   // here all other IO problems are handled
}
```

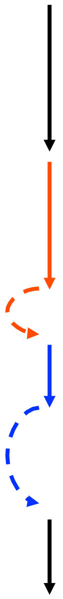
Execution flow

- open and close can generate a FileError
- read can generate a IOError

```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
  f.open();
  f.read();
  f.close();
}catch(FileError fe){
  System.out.print(“File err”);
}catch(IOError ioe){
  System.out.print(“I/O err”);
}

System.out.print("End");
```

Execution flow

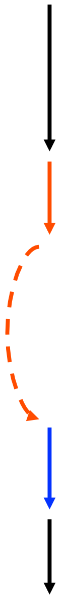
- `close` fails
- “File error” is printed
- Eventually program terminates with “End”



```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
  f.open();
  f.read();
  f.close();
}catch(FileError fe){
  System.out.print(“File err”);
}catch(IOError ioe){
  System.out.print(“I/O err”);
}

System.out.print("End");
```

Execution flow

- `read` fails
- “I/O error” is printed
- Eventually program terminates with “End”



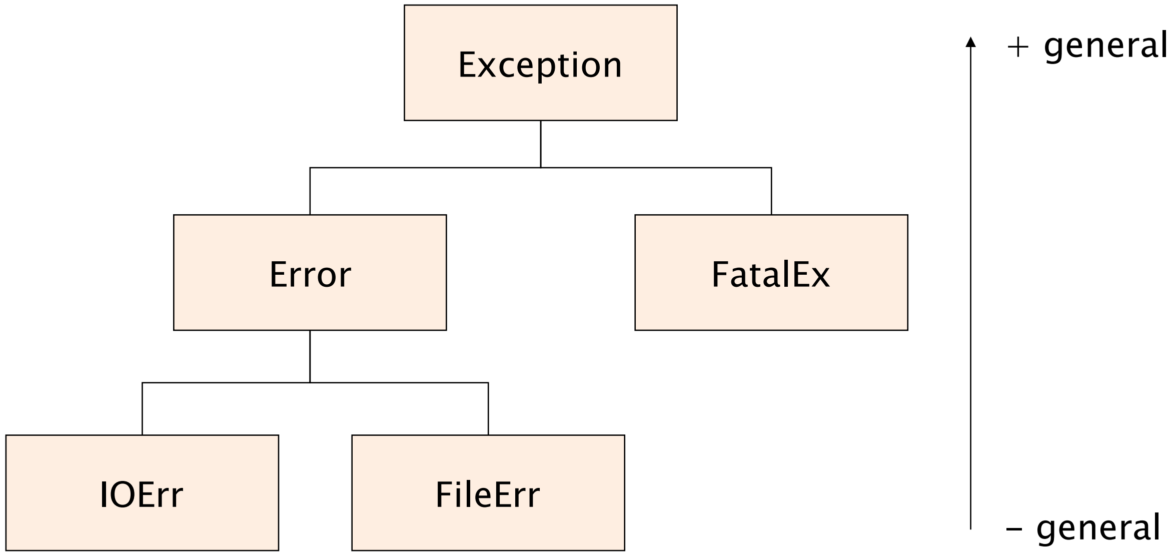
```
System.out.print("Begin");

File f = new File(“foo.txt”);
try{
  f.open();
  f.read();
  f.close();
}catch(FileError fe){
  System.out.print(“File err”);
}catch(IOError ioe){
  System.out.print(“I/O err”);
}

System.out.print("End");
```

Matching rules

- Only one handler is executed
- The more specific handler is selected, according to the exception type

- Handlers must be ordered according to their “generality”

Matching rules



Matching rules

```
class Error   extends Exception{}
class IOErr   extends Error{}
class FileErr extends Error{}
class FatalEx extends Exception{}

try{ /*…*/ }
catch(IOErr ioe){ /*…*/ }
catch(Error er){ /*…*/ }
catch(Exception ex){ /*…*/ }
```

– general

+ general

Matching rules

```
class Error   extends Exception{}
class IOErr   extends Error{}
class FileErr extends Error{}
class FatalEx extends Exception{}

try{ /*…*/ }
catch(IOErr ioe){ /*…*/ }
catch(Error er){ /*…*/ }
catch(Exception ex){ /*…*/ }
```

IOErr is generated

Matching rules

```
class Error   extends Exception{}
class IOErr   extends Error{}
class FileErr extends Error{}
class FatalEx extends Exception{}

try{ /*…*/ }
catch(IOErr ioe){ /*…*/ }
catch(Error er){ /*…*/ }
catch(Exception ex){ /*…*/ }
```

Error or FileErr is generated

Matching rules

```
class Error   extends Exception{}
class IOErr   extends Error{}
class FileErr extends Error{}
class FatalEx extends Exception{}

try{ /*…*/ }
catch(IOErr ioe){ /*…*/ }
catch(Error er){ /*…*/ }
catch(Exception ex){ /*…*/ }
```

FatalEx is generated

Nesting

- Try/catch blocks can be nested
- E.g. error handler may generate new exceptions

-
  ```
  try{ /* Do something */ }
  catch(…){
     try      { /* Log on file */ }
     catch(…){ /* Ignore */       }
  }
  ```

Generate and catch

- When calling code, which possibly raises an exception, the caller can
  - Catch
  - Propagate
  - Catch and re-throw

[1] Catch

```
class Dummy {
  public void foo(){
    try{
      FileReader f;
      f = new FileReader(“file.txt”);
    } catch (FileNotFound fnf) {
       // do something
    }
  }
}
```

[2] Propagate

```
class Dummy {

  public void foo() throws FileNotFound{
    FileReader f;
    f = new FileReader(“file.txt”);
  }

}
```

[2] Propagate (cont'd)

- Exception not caught can be propagated till main() and VM

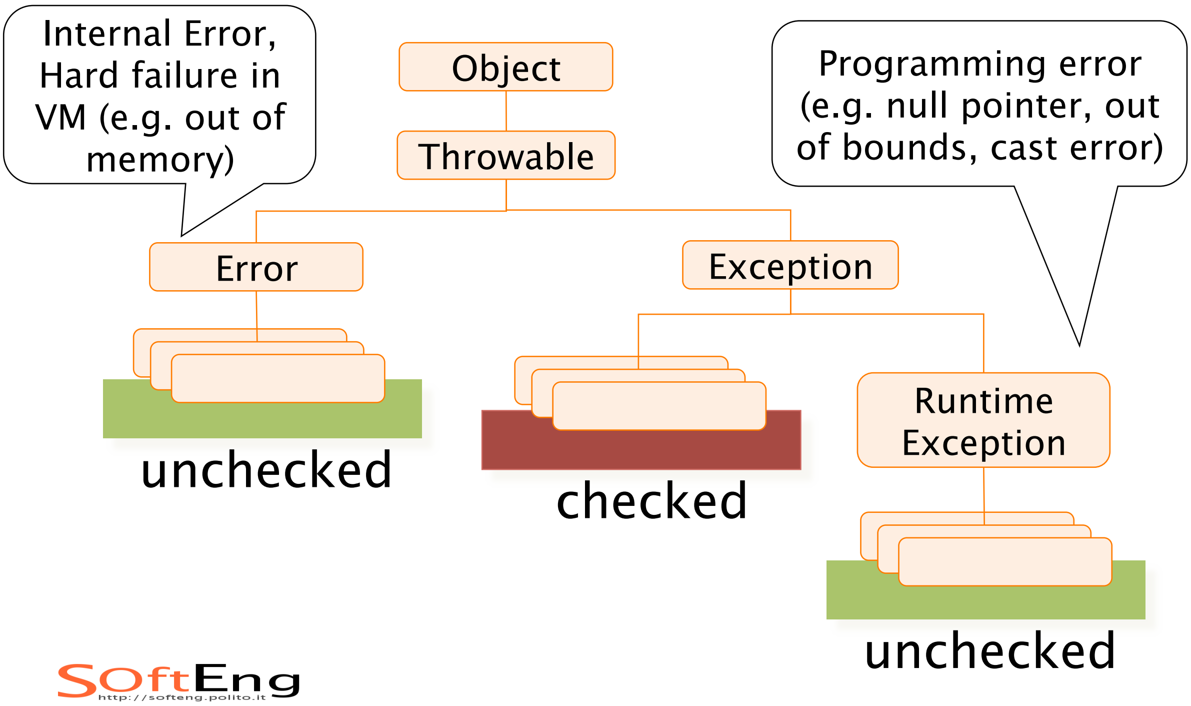
```
class Dummy {
   public void foo() throws FileNotFound {
      FileReader f = new FileReader("file.txt");
   }
}
```

```
class Program {
   public static
   void main(String args[]) throws FileNotFound {
      Dummy d = new Dummy();
      d.foo();
}
```

[3] Re-throw

```
class Dummy {
  public void foo() throws FileNotFound{
    try{
      FileReader f;
      f = new FileReader("file.txt");
    } catch (FileNotFound fnf) {
       // handle fnf, e.g., print it
       throw fnf;
    }
  }
}
```

Exceptions hierarchy


Internal Error, Hard failure in VM (e.g. out of memory)
Object
Throwable
Programming error (e.g. null pointer, out of bounds, cast error)
Error
Exception
unchecked
checked
Runtime Exception
unchecked

Exception classes – examples

- Error
  - OutOfMemoryError
- Exception
  - ClassNotFoundException
  - InstantiationException
  - NoSuchMethodException
  - IllegalAccessException
  - NegativeArraySizeException
  - EmptyStackException
- RuntimeException
  - NullPointerException

Custom exceptions

- It is possible to define new types of exceptions
- If the ones provided by the system are not enough…
- Just sub-classing Throwable or one of its descendants

Checked and unchecked

- Unchecked exceptions
  - ♦ Their generation is not foreseen (can happen everywhere)
  - ♦ Need not to be declared (not checked by the compiler)
  - ♦ Generated by JVM
- Checked exceptions
  - ♦ Exceptions declared and checked
  - ♦ Generated with “throw”

finally

- The keyword finally allows specifying actions that must be always executed
  - Dispose resources
  - Close a file

After all catch branches (if any)

```
MyFile f = new MyFile();
if (f.open(“myfile.txt")) {
   try {
      exceptionalMethod();
   } finally {
      f.close();
   }
}
```

Exceptions and loops (I)

- For errors affecting a single iteration, the try-catch blocks is nested in the loop.
- In case of exception the execution goes to the catch block and then proceed with the next iteration.

```
while(true){
   try{
     // potential exceptions
   }catch(AnException e){
     // handle the anomaly
   }
}
```

Exceptions and loops (II)

- For serious errors compromising the whole loop the loop is nested within the try block.
- In case of exception the execution goes to the catch block, thus exiting the loop.

```
try{
    while(true){
       // potential exceptions
    }
}catch(AnException e){
    // print error message
}
```

Testing exceptions

- Two main cases shall be checked:
- We expect an anomaly and therefore an exception should be rised
  - In this case the tests fails whether NO exception is detected
- We expect a normal behavior and therefore no exception should be raised
  - In this case the tests fails whether that exception in raised

Expected exception test

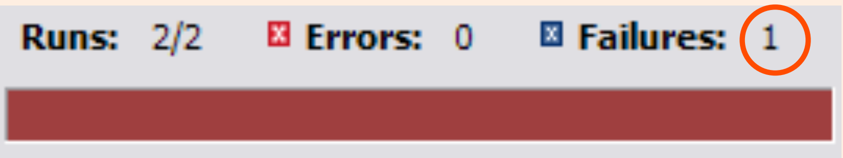
```
try{
  //  e.g. method invoked with “wrong” args
  obj.method(null);
  fail(“Methdo didn’t detected an anomaly");
}catch(PossibleException e){
  assertTrue(true); // OK
}
```

```
class TheClassUnderTest {
 public void method(String p)
       throws PossibleException
  { /*... */ }
}
```

Unexpected exception test

```
try{
  //  e.g. method invoked with right args
  obj.method(“Right Argument");
  assertTrue(true); // OK
}catch(PossibleException e){
 fail(“Method should not raise except.");
}
```

Exception → Failure

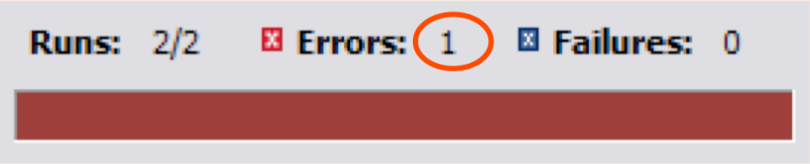


Runs: 2/2 ☒ Errors: 0 ☒ Failures: 1

Unexpected exception test

```
public void testSomething()
      throws PossibleException {
   // e.g. method invoked with right args
   obj.method(“Right Argument");
}
```

Eccezione ➔ Error



Runs: 2/2 ☒ Errors: 1 ☒ Failures: 0

License (1)

License (2)

- **4. Restrictions.**The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:
  a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(c), as requested.
  b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
  c. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.
  d. For the avoidance of doubt, where the Work is a musical composition:
    i. Performance Royalties Under Blanket Licenses. Licensor reserves the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work if that performance is primarily intended for or directed toward commercial advantage or private monetary compensation.
    ii. Mechanical Rights and Statutory Royalties. Licensor reserves the exclusive right to collect, whether individually or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions), if Your distribution of such cover version is primarily intended for or directed toward commercial advantage or private monetary compensation.
  - **Webcasting Rights and Statutory Royalties.** For the avoidance of doubt, where the Work is a sound recording, Licensor reserves the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions), if Your public digital performance is primarily intended for or directed toward commercial advantage or private monetary compensation.

License (3)

- **5. Representations, Warranties and Disclaimer**
- UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.
- **6. Limitation on Liability.** EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
- **7. Termination**
  a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
  b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

  a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
  b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
  c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
  d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.